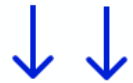
Лекция 5

Критерии готовности и критерии приёмки

SCRUM

Требования потенциальных пользователей



Product owner
(представитель заказчика)

Команда разработчиков

Скрам-мастер

Диаграмма сгорания задач

Ежедневное совещание (скрам)

Каждые 24 часа

Обзор итогов спринта

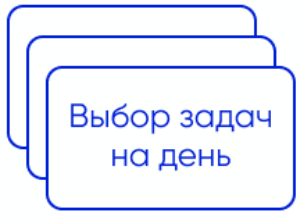
1-4 недели
Спринт

Список требований по приоритетам

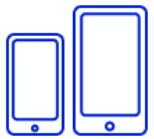
Product backlog
(техническое задание)

Команда выбирает важнейшие задачи, которые успеет решить за время спринта

Планирование спринта



Sprint backlog
(техническое задание на спринт)



Готовое ПО

Ретроспективное совещание

Спецификация требований в проектах гибкой разработки

Во многих проектах гибкой разработки на этапе выявления требований используются пользовательские истории.

Пользовательские истории накапливаются и приоритизируются в *бэглоге продукта*, который меняется на протяжении всего проекта. Крупные истории, которые охватывают значительную функциональность и которые нельзя реализовать в одной итерации, разбиваются на более мелкие, которые назначаются конкретным итерациям.

Когда команда приступает к очередной итерации, каждая история, назначенная на эту итерацию, уточняется и наполняется деталями в процессе обсуждения между владельцем продукта и командой, разрабатывающей продукт. То есть спецификация подразумевает постепенное уточнение подробностей.

В проектах гибкой разработки эти подробности часто представляются в форме критериев приемки (Acceptance Criteria), описывающих, как система должна вести себя при правильной реализации истории. Тесты истории проводятся во время итерации, в которой реализуется эта история, и в будущих итерациях в рамках регрессионного тестирования.

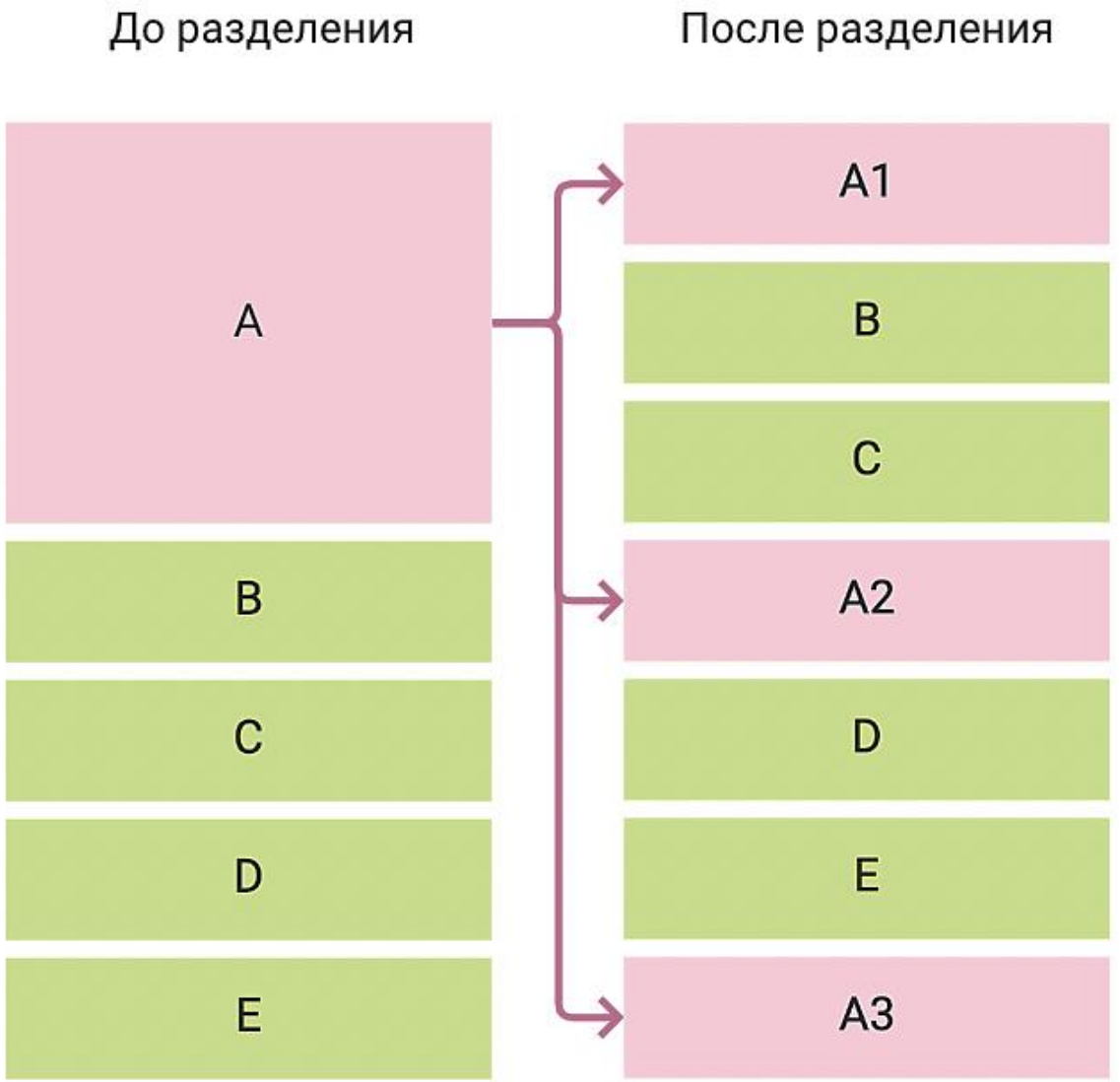
Зачем разделять крупные истории на более мелкие

1. Разделяя крупные истории на истории поменьше, вы автоматически проводите более детальный анализ задач и учитываете большее количество нюансов.
    - тем самым повышается вероятность успешного выполнения такой задачи
    - и оценка задачи будет точнее.
2. При возникновении какой-то непредвиденной ситуации, заблокированной окажется только одна история из нескольких запланированных на спринт, а не та единственная большая, которую вы еле-еле впихнули в итерацию. Аналогично, при необходимости изменений, они затронут конкретные небольшие задачи, а не одну большую целиком, и другие задачи смогут двигаться дальше.
3. Разработчик хорошо помнит код последних X часов. Чем меньше задача, тем лучше разработчик помнит написанный код, а значит быстрее исправит дефекты.
4. В конце спринта (по Скраму) история, как и любой другой элемент бэклога, должна соответствовать **Definition of Done** (DoD) – определению готовности. Команда учится завершать каждую задачу до состояния DoD, а не копить набор из десятка задач готовых на 99%.

Крупные истории разбиваются на более мелкие

5. Небольшие задачи позволяют точнее приоритизировать, и тем самым раньше поставлять самое важное и откладывать менее важное.

6. Разделение пользовательских историй помогает сохранять каждую пользовательскую историю небольшой, повышает шансы на более быструю обратной связь, что снижает риск сделать не то.



Инкременты

В соответствии с актуальным сегодня гибким подходом важное свойство процесса разработки — инкрементальность. Это значит, что продукт декомпозирован на некие части, фрагменты.

На языке организации процесса разработки фрагмент называют PBI — product backlog item, единицей продуктового бэклога. Будем использовать термин «инкремент».

Примеры инкрементов разного уровня — спринт, эпик, задача, релиз, пользовательская история... Продукт целиком — тоже инкремент, но самого высокого уровня.

Инкремент последовательно проходит несколько этапов жизни. Количество и суть этапов может различаться в зависимости от компании и команды, а также вида инкремента.

Например: Только когда аналитик закончит всю работу на своей стороне,
а QA-специалист протестирует разработанные аналитиком требования —
инкремент переходит к разработчику.

! ! ! А если на этапе, предшествующем разработке, упущено что-то важное?
Это не всегда видно невооружённым (и неопытным) взглядом, но обязательно вскроется на одном из шагов разработки.

Здесь приходит на помощь

Definition of Ready (DoR)— определение готовности.

Definition of Ready (DoR) — это набор критериев, которые определяют, когда инкремент готов к началу разработки.

Формулирование критериев DoR обычно происходит на ранних этапах планирования проекта. К процессу могут быть привлечены любые участники команды разработки, представители заказчика и пользователи продукта.

Критерии DoR должны быть максимально чёткими, понятными и достижимыми. DoR могут меняться и дорабатываться на протяжении жизни проекта по мере приобретения опыта и получения обратной связи от заинтересованных сторон.

Проверка инкремента на соответствие DoR выполняется на этапе планирования спринта. Команда обсуждает каждый инкремент (например, пользовательскую историю) и проверяет, соответствует ли он всем критериям DoR.

Инкремент, соответствующий критериям DoR, готов к началу разработки. Если инкремент не отвечает каким-то критериям — он считается не готовым и отправляется на доработку, прежде чем будет включен в производственный цикл.

В контексте UserStory DoR представляет собой некий контрольный список условий, которым должна удовлетворять US перед тем, как она может быть передана в разработку.

DoR одинаковы для всех US продукта.

Пример DoR для UserStory

- ✓ US имеет ясное и краткое описание
- ✓ Требования к инкременту протестированы
- ✓ Разработаны тест-кейсы
- ✓ Все необходимые ресурсы, спецификации и прототипы доступны команде
- ✓ Сформулированы критерии готовности (Definition of Done).
- ✓ Все зависимости от других US зафиксированы
- ✓ US приоритезирована и оценена командой
- ✓ Определены критерии успешного завершения задачи.

В контексте спринта DoR — это перечень условий, которые должны быть выполнены, чтобы команда могла начать планирование и выполнение спринта.

DoR одинаковы для всех спринтов в разработке продукта.

- ✓ Все пользовательские истории, выбранные для спринта, оценены и имеют DoR
- ✓ У команды определены цели и ожидаемые результаты для спринта
- ✓ Вся необходимая документация и дизайн доступны
- ✓ Команда имеет доступ к необходимым ресурсам, таким как серверы, тестовые данные и так далее
- ✓ Команда имеет определённый бюджет и ресурсы для выполнения спринта
- ✓ Команда имеет понимание требований продукта и его структуру
- ✓ Команда имеет определённый план спринта и понимает, как он будет реализован
- ✓ Команда имеет определённую методологию разработки, такую как Scrum или Kanban
- ✓ Команда имеет общее понимание того, какие задачи будут выполнены в течение спринта и кто за них ответственен
- ✓ Команда имеет определённую систему отслеживания прогресса и отчётности о выполнении задач в рамках спринта

Итак, инкремент перешёл к команде разработки.

! ! ! НО как понять, что работа над инкрементом завершена и готовность инкремента — 100%?

Здесь на помощь приходит

Definition of Done (DoD) – критерии «сделанности», готовности к использованию.

Примеры критериев DoD:

1. **Код**. Закончена разработка и код-ревью, код соответствует принятым в компании стандартам, отсутствуют ошибки и предупреждения компилятора (алерты). Код успешно компилируется, все составляющие продукта функционируют корректно.

2. **Тестирование**. Продукт прошёл все ручные и автотесты по кейсам, разработанным QA-специалистами. Тестами покрыт необходимый заказчику объём требований.

3. **Дизайн и интерфейс**. Интерфейс соответствует стандартам: внешним (например, гайдам мобильных платформ или основам доступности пользователям с ограниченными возможностями) и внутренним (например, фирменному стилю и дизайн-системе)

4. **Документация**. Документация полна и объясняет всё, что необходимо для использования инкремента.

5. **Интеграция**. Инкремент успешно интегрирован в общую систему или проект.

DoD Definition of Done

Для DoD актуальны те же особенности, что для DoR:

- ✓ Могут применяться как к US, так и к спринтам
- ✓ Одинаковы для всех US (спринтов) в разработке продукта
- ✓ Определяются командой до начала первого спринта
- ✓ Могут видоизменяться в процессе работы команды

DoD для US

DoD (Definition of Done) — перечень условий, которым должна соответствовать US для того, чтобы её можно было передать в эксплуатацию пользователям.

Примеры DoD для US

- ✓ Реализация US соответствует Acceptance Criteria
- ✓ Код, связанный с историей протестирован и отлажен
- ✓ Документация истории создана или обновлена
- ✓ Все необходимые одобрения истории получены
- ✓ Код, связанный с историей выложен в систему контроля версий
- ✓ Все возможные риски оценены и предусмотрены

DoD для для спринта

В контексте спринта DoD — список условий, которые должны выполняться для завершения спринта.

Примеры DoD для спринта

- ✓ Все пользовательские истории, выбранные для релиза, завершены и протестированы
- ✓ Качество продукта оценено и проверено командой QA
- ✓ Документация обновлена и актуальна
- ✓ Продукт совместим с целевой производственной средой
- ✓ Производственное окружение настроено и готово к использованию
- ✓ Продукт интегрирован с другими необходимыми системами и приложениями
- ✓ Все задачи, связанные с релизом, закрыты
- ✓ Команда имеет понимание процесса релиза и план выпуска
- ✓ Ресурсы, такие как серверы, базы данных и т.д., готовы к использованию
- ✓ Продукт протестирован и утверждён заказчиком

DoR и DoD — универсальные критерии. Как правило, они применимы ко всем инкрементам одного уровня в рамках продукта. То есть можно сформировать один набор критериев и применять их ко всем пользовательским историям.

Набор критериев DoR и DoD для задач и спринтов можно создать один раз. Их не нужно каждый раз писать заново.

С Acceptance Criteria дело обстоит иначе.

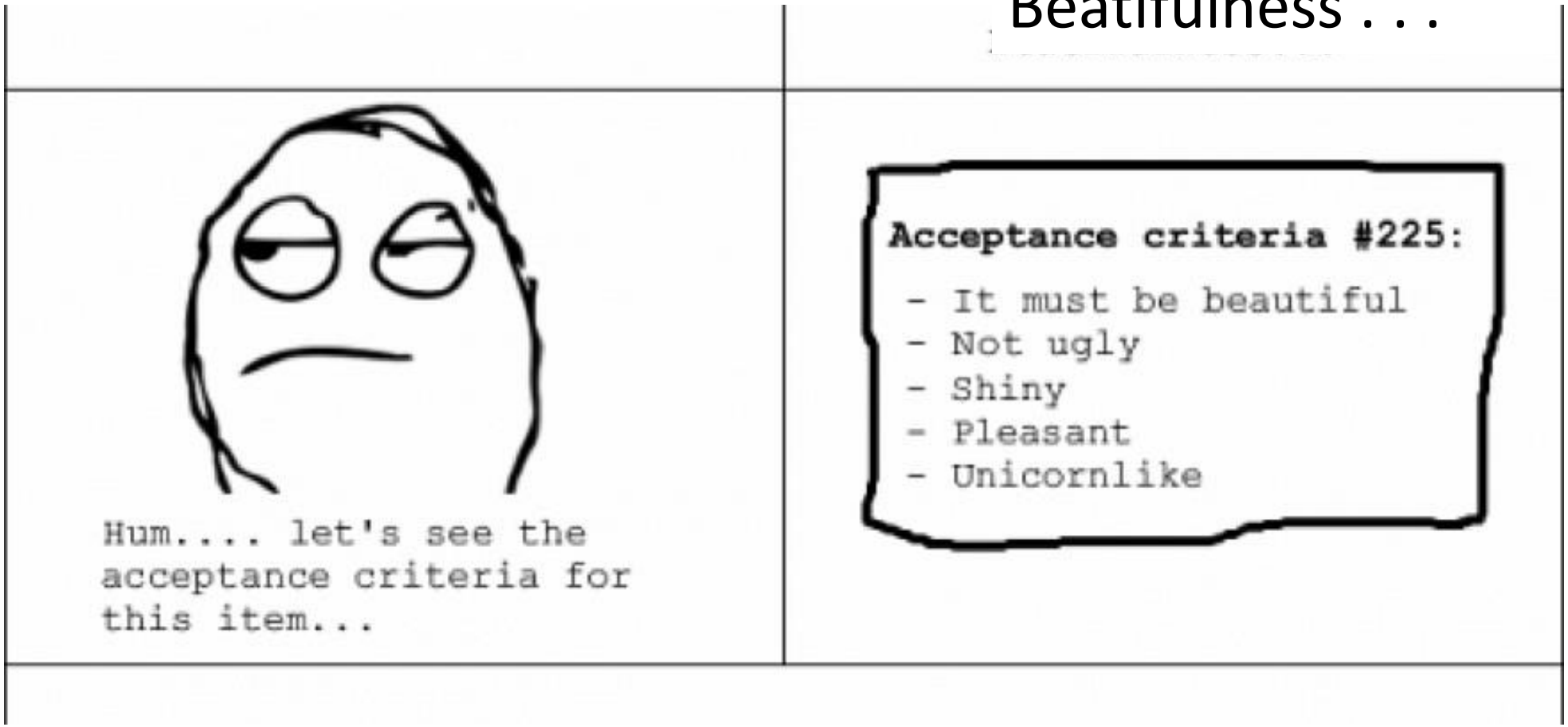
Acceptance Criteria - это набор условий и требований, которые определяют, что должен «уметь» продукт или фича, чтобы считаться успешно завершёнными.

Acceptance Criteria обычно формулируются в виде конкретных верифицируемых условий, которые должны быть выполнены. Они могут быть связаны с функциональностью, производительностью, надёжностью, пользовательским опытом и другими требованиями.

Для каждого инкремента АС будут уникальными. Они также служат основой для проведения тестирования.

Acceptance Criteria также применимы к любыми элементам бэклога: пользовательским историям, задачам, этапам, релизам и версиям.

Acceptance Criteria простыми словами :)



Подходы к написанию Acceptance Criteria

✓ Свод правил (шаблон чеклиста)

Свод правил включает в себя описание требуемого функционала, ограничения по производительности, безопасности и другие.
В этом случае критерии оформляются в виде обычного списка.

✓ Сценарно-ориентированный подход (шаблон Given/When/Then)

Суть сценарно-ориентированного подхода заключается в описании поведения программы через пользовательские сценарии использования.
Для такого описания используются специальные языки.

Свод правил

Пример:

***Как** пользователь*
***Я хочу** искать товар в пределах установленного диапазона стоимости*
***Чтобы** найти подходящие мне товары в моей ценовой категории*

Формат истории позволяет понять задачу, в ней указана роль, функционал и польза от его реализации. Мы обсудили и оценили User Story с командой, определили, что US соответствует INVEST-критериям: она не имеет зависимостей, достаточно небольшая и тестируемая.

Однако представленного описания недостаточно для передачи User Story в разработку. Поэтому мы должны задать себе вопрос: «Какие проверки должна выполнить команда перед передачей функции заказчику?»

Задаем вопросы заказчику

1. *Какой диапазон стоимости товаров будет доступен для поиска?*
2. *Будет ли у пользователя возможность указать точную цену или только минимальную и максимальную цену?*
3. *Нужна ли функция автодополнения при вводе диапазона цен, чтобы пользователи могли быстро выбрать подходящий диапазон?*

Вопросы имеют большое значение для пользовательского опыта и являются ключевыми для понимания ожиданий от функции.

4. *Какие ошибки или нежелательное поведение должны быть обработаны, если пользователь вводит неправильный диапазон цен?*
5. *Нужно ли кэшировать результаты поиска, чтобы ускорить поиск при повторном запросе с тем же диапазоном цен?*
6. *Как будет осуществляться поиск товаров в заданном диапазоне цен? Будет ли это происходить автоматически при вводе диапазона, или пользователю нужно будет нажать кнопку для начала поиска?*

Тоже важны для пользователя, но, если заказчик не готов на них ответить, эти аспекты стоит решить в команде..

7. *Каким образом будут отображаться результаты поиска товаров в заданном диапазоне цен?*
8. *Какие ограничения на количество результатов поиска?*
9. *Будут ли в поиске учитываться товары, которых нет в наличии?*
10. *Будут ли в поиске учитываться товары, ожидающие поступления в ближайшее время?*

Вопросы относятся к функционалу, который может улучшить опыт пользователя и производительность системы, но не являются обязательными.

Команда

Какой диапазон стоимости товаров будет доступен для поиска?

Заказчик

Нам нужно, чтобы пользователь мог указывать диапазон цен самостоятельно, но мы также хотим предоставить предустановленные фильтры для цен, например, дешевые товары до 50 рублей или дорогие товары свыше 5 000 рублей.

Команда

Будет ли у пользователя возможность указать точную цену или только минимальную и максимальную цену?

Заказчик

Хотелось бы, чтобы пользователь мог указывать как точную цену, так и диапазон цен. Можно использовать бегунок или вводить диапазон цены с клавиатуры

Нужна ли функция автодополнения при вводе диапазона цен, чтобы пользователи могли быстро выбрать подходящий диапазон?

Заказчик

Да, мы бы хотели, чтобы функция поддерживала автодополнение, чтобы пользователи могли быстро выбрать подходящий диапазон. Причем автодополнение должно учитывать категорию товара. Например, для категории "Одежда" диапазон может быть от 100 до 50 000 рублей, а для категории "Электроника" - от 50 до 5 000 000 рублей.

Команда

Какие ошибки или нежелательное поведение должны быть обработаны, если пользователь вводит некорректный диапазон цен, например, указывает минимальную цену больше максимальной?

Заказчик

Если пользователь вводит неправильный диапазон цен, мы должны сообщить ему об этом и запросить правильный диапазон.

Команда

Будем ли мы кэшировать результаты поиска? Это позволит при повторном запросе с тем же диапазоном цен ускорить работу приложения и уменьшить нагрузку на сервер. Но нужно учитывать, что в таком случае возможна устаревшая информация в кэше, если цены на товары изменились с момента последнего поиска.

Заказчик

Мы хотим сохранять результаты поиска в кэше на сервере и использовать их при следующем запросе. Но при смене стоимости товара мы хотим, чтобы результат поиска был актуальным.

Команда

Каким образом будут отображаться результаты поиска товаров в заданном диапазоне цен?

Команда

Какие ограничения на количество результатов поиска?

Заказчик

Мы предпочитаем отображать результаты поиска в виде строк, по 10 штук на странице, товары расположить в порядке возрастания цены с возможностью сортировки по цене или другим параметрам.

Команда

Будут ли в поиске учитываться товары, которых нет в наличии?

Команда

Будут ли в поиске учитываться товары, ожидающие поступления в ближайшее время?

Заказчик

Мы не хотим выводить товары, которых нет в наличии. Но хотим показывать товары, подвоз которых ожидается в ближайшее время.

Оформляем **Acceptance Criteria** в виде свода правил с учётом ответов заказчика:

1. *Пользователь может установить свой диапазон цены поиска при помощи бегунка, где изначальный диапазон цен соответствует минимальной и максимальной ценам товаров в выбранной категории или группы категорий.*
2. *Пользователь может установить диапазон цены с клавиатуры.*
3. *Для активации поиска пользователь нажимает кнопку «Найти».*
4. *Если пользователь вводит некорректный диапазон цен, система должна выдавать сообщение об ошибке и не показывать пустой список товаров. В приложении — примеры экранных форм и нотификаций об ошибках.*
5. *Результаты поиска должны быть точными и соответствовать установленному диапазону цен, по умолчанию отображаются в порядке возрастания цены.*

Оформляем **Acceptance Criteria** в виде свода правил с учётом ответов заказчика:

6. *Результаты поиска разбиваются по 10 штук на странице, обеспечивается переход на предыдущие или следующие страницы при помощи ссылок.*
7. *Если результаты поиска были закэшированы, то они должны быть конкретными и актуальными, с учётом изменений в ценах на товары.*
8. *Функция поддерживает автодополнение, которое учитывает категорию товара, чтобы пользователи могли быстро выбрать подходящий диапазон цен.*
9. *Для каждой категории товаров предусмотреть предустановленные фильтры для цен: «дешёвые товары», «дорогие товары» и прочее. Варианты — в зависимости от категории — в приложении.*
10. *Результаты поиска исключают товары, которых нет в наличии и включают товары, подвоз которых ожидается.*

Дополняем список

Нефункциональные требования также являются важными для успешной разработки и работы ПО. Заказчики могут иметь определённое представление о том, как ПО должно работать, но, как правило, команда разработки лучше знает его возможности и ограничения.

Важно договориться о том, какие требования к надёжности, доступности, производительности будут у разрабатываемого ПО. Например, заказчик может формулировать требования к информационной безопасности таким образом: ПО должно иметь защиту от взлома и несанкционированного доступа.

Обсуждаем полученные от заказчика ответы и список критериев приёмки с владельцем продукта и командой разработки. В рамках обсуждения **добавляем в список следующие ограничения**:

11. Время поиска товаров должно быть не более 3 секунд.

12. Функция поиска должна быть защищена от SQL-инъекций и атак на XSS.

13. Проверка входных данных должна быть осуществлена, чтобы предотвратить возможность межсайтовой подделки запросов (CSRF).

Сценарно-ориентированный подход при описании критериев приёмки

Сценарно-ориентированный подход представляет собой критерии приемки, описанные по шаблону **Given/When/Then**.

- **Given** заданное предварительное условие
- **When** я выполняю какое-то действие
- **Then** я ожидаю какой-то результат

Подход получил своё развитие при появлении **BDD** (Behavior Driven Development — «разработка через поведение») — метода разработки, когда сначала пишутся приёмочные тесты, а потом код).

Преимущества сценарно-ориентированного подхода

- ✓ Простота в написании и понимании как для разработчиков, так и для других заинтересованных сторон. Это облегчает взаимодействие между командой разработки и заказчиком.
- ✓ Ориентация на требования разрабатываемого продукта гарантирует, что процесс разработки будет основываться на потребностях конечных пользователей.
- ✓ Возможность повторного использования кода благодаря тому, как написаны эти сценарии. Это позволяет экономить ресурсы и использовать части этого кода при создании других тестов.

Использование Gherkin для сценарно-ориентированного подхода описания критериев приемки.

Gherkin — это сценарно-ориентированный язык, который легко читается бизнесом и используется для описания функциональности программного обеспечения. Использование Gherkin позволяет сократить разрыв между бизнесом и разработчиками

***Как** пользователь*
***Я хочу** искать товар в пределах установленного диапазона стоимости*
***Чтобы** найти подходящие мне товары в моей ценовой категории*

```
Feature:  Фильтрация товаров по цене
Scenario: Поиск товаров по диапазону цен - базовый вариант
   Given  Я нахожусь на главной странице магазина
   When   Я ввожу "1000" в поле "от"
   And    Я ввожу "5000" в поле "до"
   And    Я нажимаю кнопку "Найти"
   Then   Я вижу список товаров со стоимостью в указанном диапазоне
   But    Я не вижу товары, которых нет в наличии
```

```
Feature:  Фильтрация товаров по цене
Background:
   Given Я авторизованный пользователь
   And   Я нахожусь на главной странице магазина
```

Сценариев может быть несколько

```
 Scenario: Поиск товаров по диапазону цен - базовый вариант
    When   Я ввожу "1000" в поле "от"
    And    Я ввожу "5000" в поле "до"
    And    Я нажимаю кнопку "Найти"
    Then   Я вижу список товаров со стоимостью в указанном диапазоне
    But    Я не вижу товары, которых нет в наличии
```

```
Scenario: Поиск товаров по диапазону цен из листа ожидания
   When   Я ввожу "1000" в поле "от"
   And    Я ввожу "5000" в поле "до"
   And    Я выбираю вариант “До двух дней” из выпадающего
          списка “Могу подождать до”
   And    Я нажимаю кнопку "Найти"
   Then   Я вижу список товаров со стоимость в указанном диапазоне
   And    товары  в статусах “в Наличии” и “Подвоз до 2х дней”
```

Вывод

Acceptance Criteria (AC) решают множество задач, главная из которых — избежать двусмысленности и обеспечить понимание того, что должно быть достигнуто в результате работы над User Story.

- ✓ **Acceptance Criteria как свод правил** имеет свободный формат и может включать ограничения на функциональность, производительность, безопасность или любые другие требования, которые могут быть сформулированы в виде списка.
- ✓ **Сценарно-ориентированный подход** — описание поведения программы через сценарии использования. Мы разобрали на примере как можно писать сценарии на языке Gherkin.

Сравнительная таблица: DoR, DoD, AC (в контексте US):

Definition of Ready (DoR) Готовность (подготовленность) User Story к передаче в работу	**Definition of Done (DoD)** Готовность User Story к передаче в эксплуатацию.	**Acceptance Criteria (AC)** Критерии приёмки результата реализации US
DoR — контрольный список команды, позволяющий убедиться, что у вас есть всё необходимое для начала работы над User Story.	DoD — контрольный список команды, обеспечивающий ясность и однородность в определении всеми членами команды того, что работа над User Story полностью завершена.	AC — список условий, которые должны выполняться для US, чтобы она удовлетворяла потребностям пользователя и соответствовала ожиданиям заказчика.
Какие		
Одинаковые для всех User Stories продукта.		Уникальные для каждой User Story.

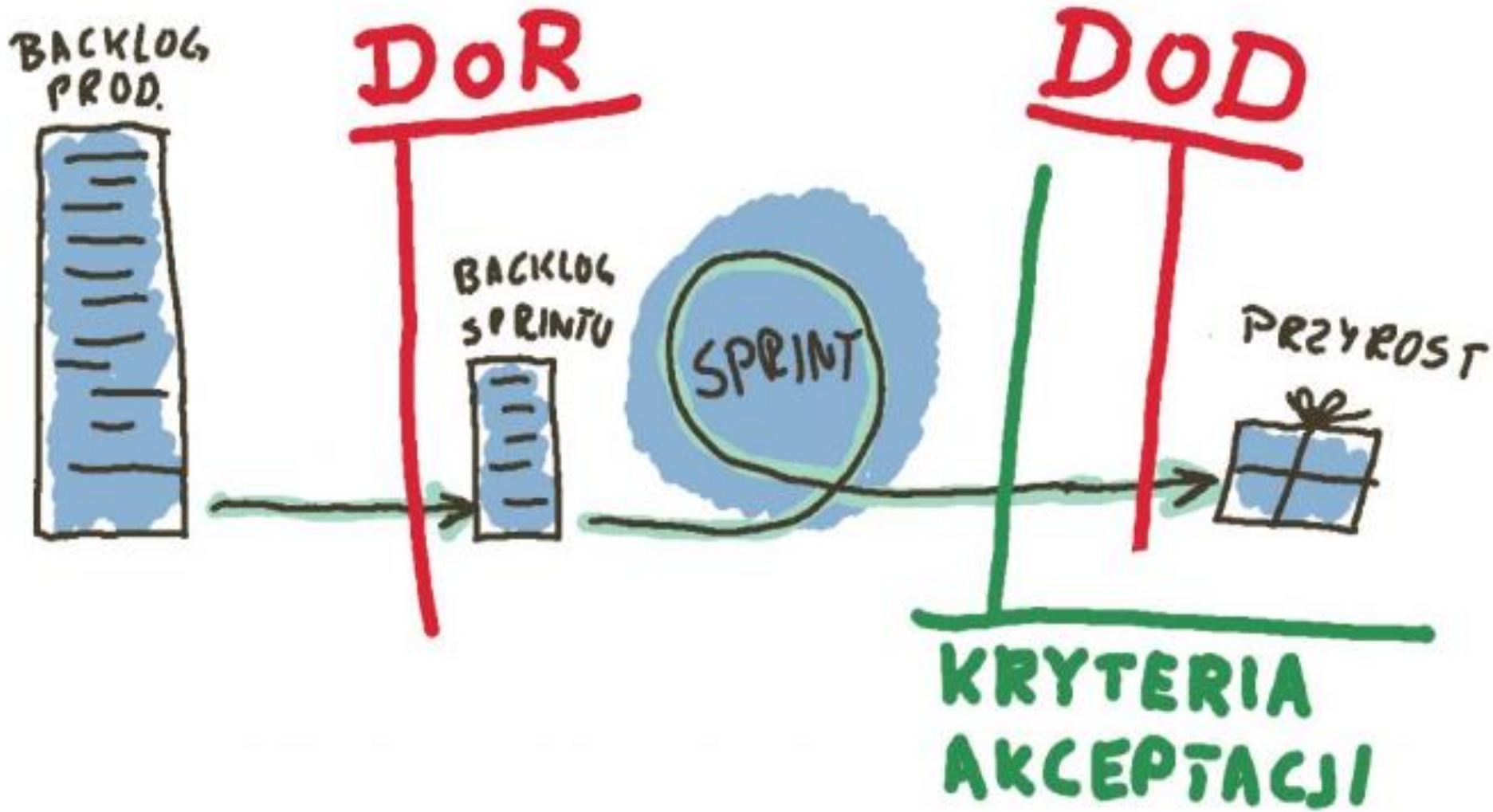
Связь критериев готовности и критериев приёмки с этапами жизненного цикла US графически можно изобразить так:

Этапы жизненного цикла User Story

Этап	Описание этапа
Определение требований и запись US	Аналитики определяют потребности пользователей, формулируют конкретные требования к системе и записывают US с использованием определённого формата (название, описание, приоритет, **критерии приёмки** и т.д.)
Оценка US	Вся команда оценивает сложность и затраты, условия и готовность к реализации каждой US. При этом не требуется оценивать сразу все US в бэклоге — достаточно оценить самые приоритетные. Именно здесь команде нужны DoR — переход US на следующий этап невозможен, если она не соответствует критериям DoR.
Планирование итераций	Вся команда планирует реализацию US в конкретные итерации продукта.

Этапы жизненного цикла User Story в Agile

Этап	Описание этапа
Реализация US	Команда разработки реализует US в соответствии с определёнными критериями приёмки — Acceptance Criteria, уникальными для каждой US.
Проверка и приёмка US	US проверяется на готовность к сдаче конечному пользователю, а также на соответствие заданным критериям приёмки и, если они успешно проходят проверку, то считаются завершёнными. Здесь команде нужны AC и DoD. US не может быть объявлена готовой без соответствия AC и не может перейти на следующий этап, если она не соответствует DoD.
Передача US в эксплуатацию	Когда реализованная US соответствует DoD, она включается в состав инкремента продукта и передаётся в эксплуатацию пользователю.


BACKLOG PROD.
DoR
BACKLOG SPRINTU
SPRINT
DOD
PRZYROST
KRYTERIA AKCEPTACJI